Lecture 9

Sorting and Strings

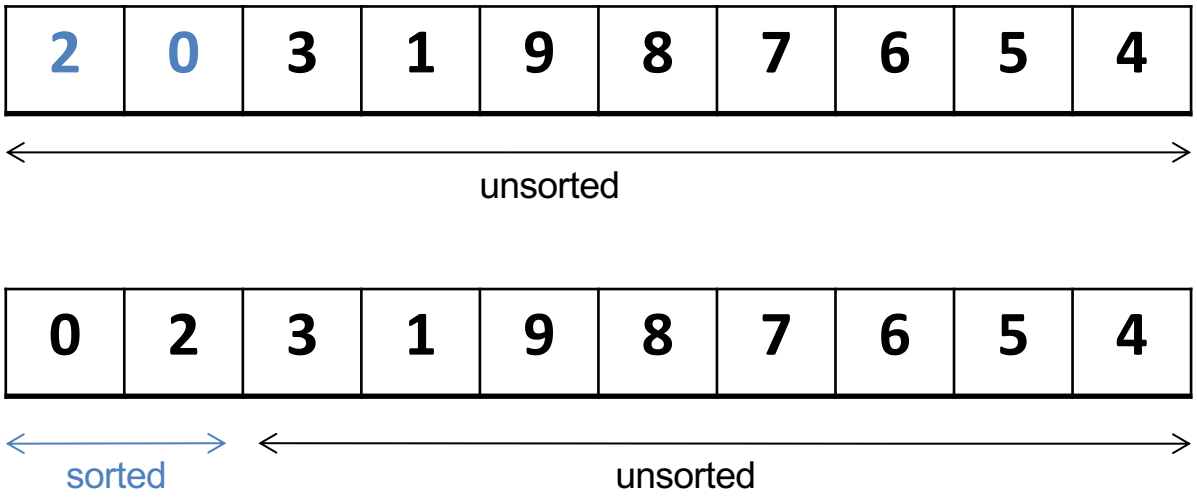
Announcement

- Lab this week (Feb 7)
  - Available on Moodle at specified lab time
  - Upload the right file (specified in lab handout)
  - Moodle crashes (screenshot with timestamp and contact IT)
- Midterm next week
  - Regular class time and all in Moodle
  - Multiple choice questions
  - Coding questions like the lab
  - Feb 15 or 16: depending on section
  - Feb 12 or 13: no class, but office hours virtually, don't ask exam for question
  - Feb 14: no lab

Insertion Sort

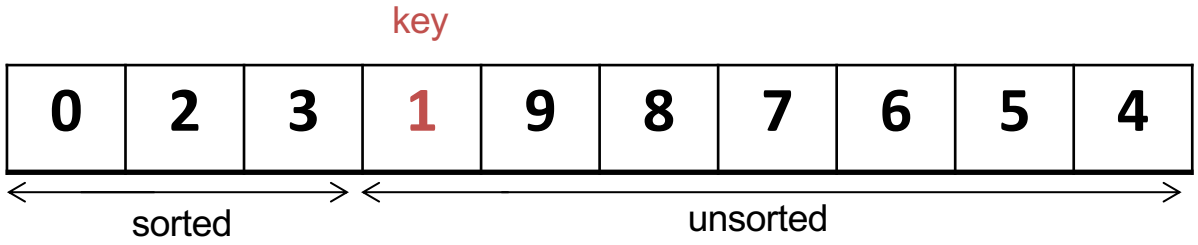
The array is split into two parts, a sorted part and an unsorted part

Values from the unsorted part are picked and sorted properly into the sorted part

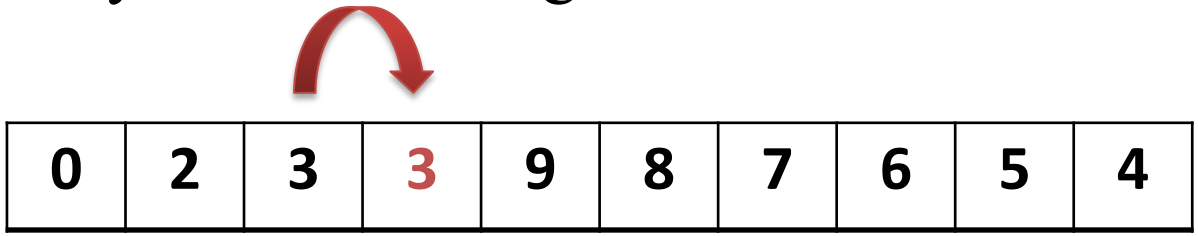


Insertion Sort (cont.)

Let's jump to the third loop iteration, the element in orange, known as the key compares itself with its left most element

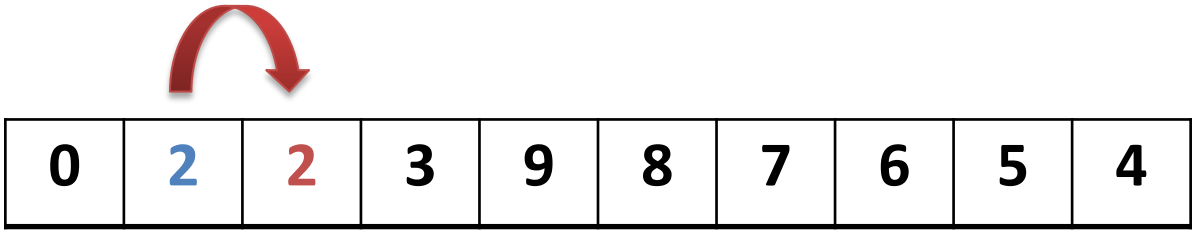


If the left most element is greater then the key's value, then we update key's index to the greater value



Insertion Sort (cont.)

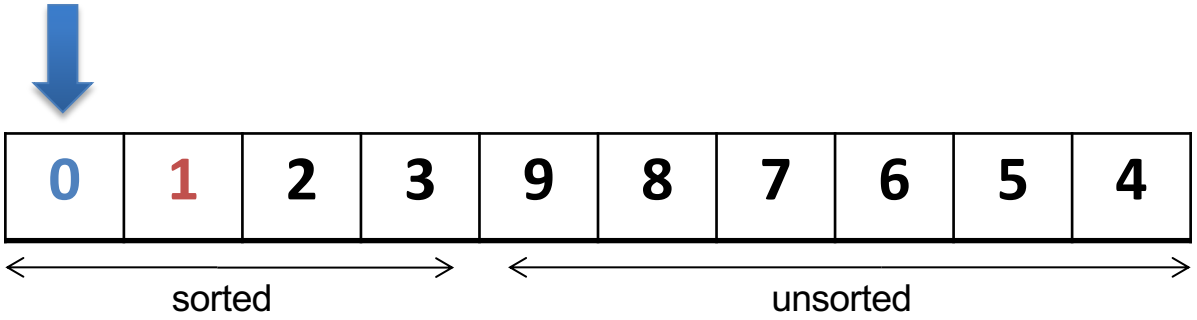
We now check the next left most element and compare it to key's value which is still 1



Since next left most element is still greater than key's value, we update the element right of it

Insertion Sort (cont.)

Again, we check the next left most element and compare it to key's value

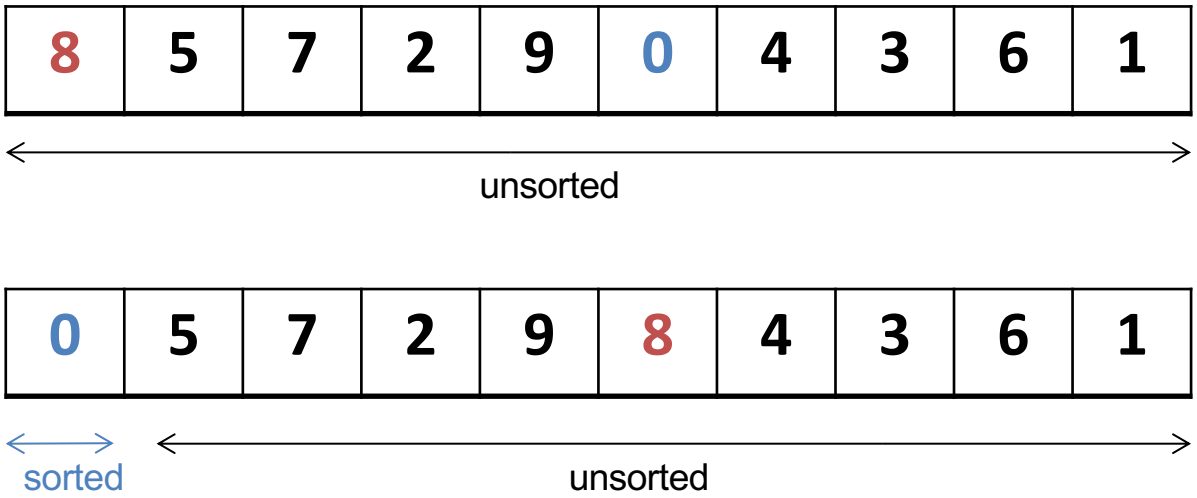


Since next left most element is less than key's value, we update the element right of it to keys value

Selection Sort

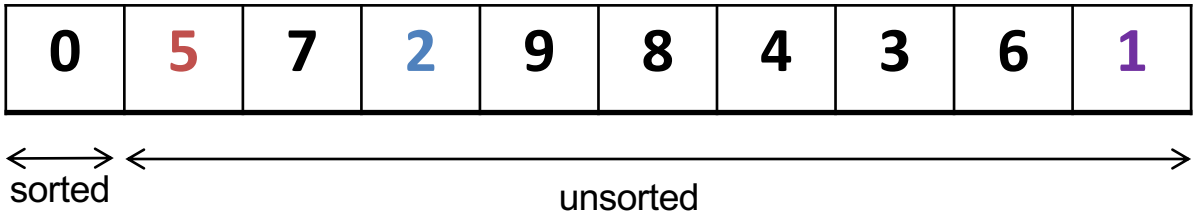
Like Insertion sort the array is split into the sorted and unsorted parts

Each iteration the lowest element from the unsorted part gets put in front of the unsorted part swapping values with the iteration index



Selection Sort (cont.)

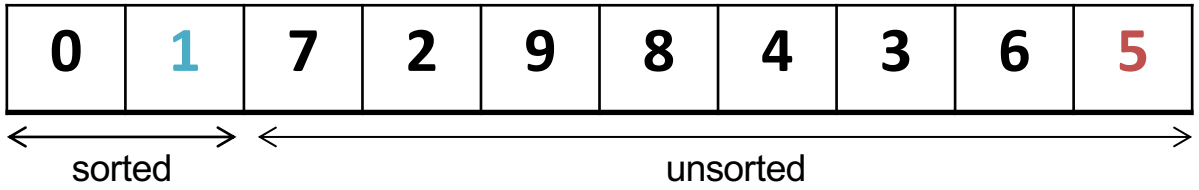
We start each iteration with the current index being set as the 'minimum' (*min*)
In a nested loop it iterates through the unsorted part replacing the *min* value with any smaller



Here we start by setting minimum to index 1
Then since 2 is smaller then 5, minimum gets set to index 3
Then since 1 is smaller then 2, minimum gets set to index 9

Selection Sort (cont.)

Index 9 is then swapped with the parent loop index, which is 1



And the pattern continues with index 2

String: Text as a Value

- String are quoted characters
  - 'abc d' (Python prefers)
  - "abc d" (most languages)
- How to write quotes in quotes?
  - Delineate with "other quote"
  - **Example**: "Don't" or '6" tall'
  - What if need both " and ' ?
- **Solution**: escape characters
  - Format: \ + letter
  - Special or invisible chars

Char	Meaning
\'	single quote
\"	double quote
\n	new line
\t	tab
\\	backslash

```
>>> x = 'I said: "Don\'t"'
>>> print(x)
I said: "Don't"
```

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is s[3:6]?

A: 'lo a'
B: 'lo'
C: 'lo '
D: 'o '
E: I do not know

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is s[3:6]?

A: 'lo a'
B: 'lo'
C: 'lo ' **CORRECT**
D: 'o '
E: I do not know

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is s[:4]?

A: 'o all'
B: 'Hello'
C: 'Hell'
D: Error!
E: I do not know

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is s[:4]?

A: 'o all'
B: 'Hello'
C: 'Hell' **CORRECT**
D: Error!
E: I do not know

Other Things We Can Do With Strings

- **Operation** in: s_1 in s_2
  - Tests if s_1 "a part of" s_2
  - Say s_1 a *substring* of s_2
  - Evaluates to a bool
- **Examples**:
  - s = 'abracadabra'
  - 'a' in s == True
  - 'cad' in s == True
  - 'foo' in s == False

- **Function** len: len(s)
  - Value is # of chars in s
  - Evaluates to an int
- **Examples**:
  - s = 'abracadabra'
  - len(s) == 11
  - len(s[1:5]) == 4
  - s[1:len(s)-1] == 'bracadabr'

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text

    # Start of middle third

    # End of middle third

    # Get the text


    # Return the result
    return result
```

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text

    # Start of middle third

    # End of middle third

    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text

    # Start of middle third

    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text


    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text
    size = len(text)
    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Defining a String Function

```
>>> middle('abc')
'b'
>>> middle('aabbcc')
'bb'
>>> middle('aaabbbccc')
'bbb'
```

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text
    size = len(text)
    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Not All Functions Need a Return

```
def greet(n):
    """Prints a greeting to the name n

    Parameter n: name to greet
    Precondition: n is a string"""
    print('Hello '+n+'!')
    print('How are you?')
```

Note the difference

Displays these strings on the screen

No assignments or return
The call frame is **EMPTY**

Procedures vs. Fruitful Functions

Procedures

- Functions that **do** something
- Call them as a **statement**
- Example: greet('Walker')

Fruitful Functions

- Functions that give a **value**
- Call them in an **expression**
- Example: x = round(2.56,1)

Historical Aside

- Historically "function" = "fruitful function"
- But now we use "function" to refer to both

Print vs. Return

Print

- Displays a value on screen
  - Used primarily for **testing**
  - Not useful for calculations

```
def print_plus(n):
    print(n+1)
>>> x = print_plus(2)
3
>>>
```

Return

- Defines a function's value
  - Important for **calculations**
  - But does not display anything

```
def return_plus(n):
    return (n+1)
>>> x = return_plus(2)
>>>
```

Print vs. Return

Print

- Displays a value on screen
  - Used primarily for **testing**
  - Not useful for calculations

```
def print_plus(n):
    print(n+1)
>>> x = print_plus(2)
3
>>>
```

x

Nothing here!

Return

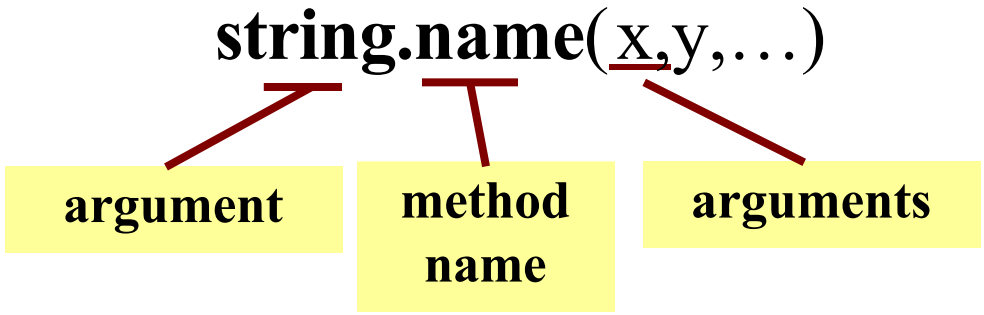
- Defines a function's value
  - Important for **calculations**
  - But does not display anything

```
def return_plus(n):
    return (n+1)
>>> x = return_plus(2)
>>>
```

x 3

Method Calls

- Methods calls are unique (right now) to strings
  - Like a function call with a "string in front"
- **Method calls** have the form



- The string in front is an **additional** argument
  - Just one that is not inside of the parentheses
  - **Why?** Will answer this later in course.

Example: upper()

- upper(): Return an upper case **copy**

```
>>> s = 'Hello World'
>>> s.upper()
'HELLO WORLD'
>>> s[1:5].upper()     # Str before need not be a variable
'ELLO'
>>> 'abc'.upper()      # Str before could be a literal
'ABC'
```

- Notice that *only* argument is string in front

Examples of String Methods

- s_1.index(s_2)
  - Returns position of the *first* instance of s_2 in s_1
- s_1.count(s_2)
  - Returns number of times s_2 appears inside of s_1
- s.strip()
  - Returns copy of s with no white-space at *ends*

```
>>> s = 'abracadabra'
>>> s.index('a')
0
>>> s.index('rac')
2
>>> s.count('a')
5
>>> s.count('x')
0
>>> '  a b '.strip()
'a b'
```

Examples of String Methods

- s_1.index(s_2)
  - Returns position of the *first* instance of s_2 in s_1
- s_1.count(s_2)
  - Return
    s_2 appe
- s.strip()
  - Returns copy of s with no white-space at *ends*

```
>>> s = 'abracadabra'
>>> s.index('a')
0
                    'rac')
                     a')
5
>>> s.count('x')
0
>>> '  a b '.strip()
'a b'
```

See Lecture page for more